VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce semaforů v obrazech

Traffic Light Detection in Images

2018 Moravec Vojtěch

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Vojtěch Moravec**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Detekce semaforů v obrazech Traffic Light Detection in Images
Jazyk vypracování:	čeština

Zásady pro vypracování:

Inteligentní asisteční systémy v dopravě se stávají běžným vybavením moderních vozidel. Jedním z takových asistenčních systémů je detekce semaforů a rozpoznání jejich stavu. Tento systém může varovat řidiče v případě jízdy na červenou nebo může být využit při vývoji autonomních vozidel. Cílem této bakalářské práce je vytvořit program pro detekci a rozpoznání stavu semaforů na základě analýzy obrazu z kamery umístěné ve vozidle.

Ve své práci proveďte:
1. Popište zadaný problém.
2. Analyzujte řešení a popište potřebnou teorii.
3. Proveďte vlastní implementaci řešení a její testování v reálných podmínkách.
4. Zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] R.C. Gonzalez, R.E. Woods: Digital Image Processing (3rd Edition), 2006, ISBN: 013168728X
[2] E. Sojka: Digitální zpracování a analýza obrazů, 2000, ISBN 80-7078-746-5
[3] M.B. Jensen, M.P. Philipsen, A. Møgelmose, T.B. Moeslund and M. M. Trivedi, "Vision for Looking at Traffic Lights: Issues, Survey, and Perspectives," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 7, pp. 1800-1815, 2016

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michael Holuša**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30.04.2018

Moravec Vojtěch

Rád bych to tomto místě poděkoval vedoucímu bakalářské práce, Ing. Michaelu Holušovi, za vstřícný přístup, ochotu a čas věnovaný k pomoci na této práci. Dále bych chtěl poděkovat Ing. Radovanovi Fuskovi Ph.D. a Ing. Janu Gaurovi Ph.D. za jejich rady.

Abstrakt

Cílem této práce je vytvoření programu, který bude schopen detekovat semafory v obrazech, přesněji jejich nalezení a následné zjištění stavu. Bude využito metody, která je v současnosti považována za nejlepší pro nalezení objektů v obrazech a to detekce pomocí konvolučních neuronových sítí. V první části práce budou popsány základní charakteristiky konvolučních neuronových sítí, vrstvy, ze kterých se tyto sítě skládají a jak vůbec tato metoda detekce funguje. Dále popíšeme získaní datasetu, který je potřeba pro trénování těchto sítí a následné připravení tohoto datasetu. V poslední části rozebereme vlastní implementaci detektoru, porovnáme testované architektury konvolučních neuronových sítí a otestujeme správnost detektoru na datech z reálného světa.

Klíčová slova: detektor, neuronové sítě, konvoluční neuronové sítě, konvoluce, Dlib, MMOD, semafor, dataset, analýza obrazu

Abstract

The purpose of this thesis is to create an application, which can detect traffic lights in images, more precisely find the traffic light in image and determine its state. To create this detector we will use convolutional neural networks, which is *state–of–the–art* method for detecting objects in images. In first part, we will describe basic principles of convolutional neural networks, layers, which build the network and how this method works as whole. Next, we will describe how we obtained our dataset, which is required for training of these networks and how we prepared that dataset. In last part, we will go through our implementation, we will compare different tested convolutional neural network architectures and finally, we will test our detector on real–life data.

Key Words: detector, neural networks, convolutional neural networks, convolution, Dlib, MMOD, traffic light, dataset, image analysis

Obsah

Seznam použitých zkratek a symbolů

CNN	–	Konvoluční neuronové sítě
CPU	–	Procesor
GPU	–	Grafická karta
MMOD	–	Max margin object detection

Seznam obrázků

Seznam tabulek

1 Úvod

V současnosti je velmi aktivní vývoj autonomních vozidel a stále více časté jsou inteligentní asistenční systémy v moderních vozidlech. Jedním z problémů, který musí být vyřešen pro obě tyto kategorie je detekce semaforu a rozpoznání jeho stavu. Pro autonomní vozidla je tento systém kritický. Určuje, zda má vozidlo začít zpomalovat, zcela se zastavit nebo jestli může pokračovat v jízdě. Pro moderní automobily může systém kontrolovat řidiče, upozornit ho na jízdu na červenou nebo sám začít zpomalovat vozidlo. Dalším fenoménem této doby, v oblasti informačních technologií, jsou neuronové sítě, lepé řečeno řešení různých problému pomocí neuronových sítí. Neuronové sítě se v současnosti dostávají skoro do všech oborů informačních technologií a asi nejvíce rozšířené jsou v oboru analýzy obrazu, ale jejich uplatnění je mnohem širší. Jako příklad můžeme uvést analýzu řečí, automatické překlady mezi různými jazyky, předpovídání vývoje akcií na akciovém trhu, předpovídání vývoje ceny nemovitostí v závislosti na historických a aktuálních datech, nebo v neposlední řadě umělá inteligence v počítačových hrách. Neuronové sítě slaví stále více úspěchů a daří se jim překonávat tradiční metody i člověka v situacích, ve kterých by to nepředpokládal, například poražení člověka v čínské deskové hře "Go". Obecně by se dalo říci, že neuronové sítě jsou použity všude tam, kde neznáme lepší způsob řešení daného problému a proto řešení necháme na nich. V této práci spojíme obojí dříve zmíněné, neuronové sítě a autonomní vozidla, budeme se zabývat detekcí semaforů v obrazech a prozkoumáme řešení založené na detekci pomocí konvolučních neuronových sítí, které se v řadě jiných aplikací v oboru analýzy obrazu ukázali jako nejlepší možný způsob řešení. Popíšeme základní funkčnost neuronových sítí, budou porovnány jejich 2 různé architektury a kroky potřebné k vlastnímu využití této metody analýzy obrazu. Dále budou prozkoumány různé způsoby detekce stavu semaforu a porovnány jejich výhody a nevýhody. Nakonec budou všechny prozkoumané způsoby řešení otestovány a ohodnoceny na reálných datech získaných z kamery umístěné ve vozidle.

2 Neuronové sítě

Lidé se snaží už po dlouhou dobu vytvořit počítačový program, co nejvíce podobný našemu mozku v tom smyslu, že umí myslet a učit se ze svých chyb. Tyto programy spadají do skupiny umělé inteligence. Zatím nejlepším kandidátem na vývoj umělé inteligence jsou neuronové sítě, inspirovány přírodou a hlavně pak lidským mozkem, jehož nejdůležitější vlastností je schopnost učení. Tuto vlastnost se snaží tyto sítě co nejlépe napodobit. Tvrzení o nejlepším kandidátovi je založeno na úspěších v porovnání s ostatními metodami a i v porovnání s člověkem, kde ho umělá inteligence porazila. V případě této práce chceme použít neuronovou síť k detekci semaforů v obrazech, dalo by se říct, že po ní chceme, aby se naučila rozeznávat semafory od ostatních objektů v obrazu.

2.1 Historie

S neuronovými sítěmi nebo metodami jim podobným se člověk potýká už celkem dlouhou dobu. První modely, které ještě nebyly schopny učení se datují do roku 1943. První modely již s umělými neurony byly představeny v 60. letech 20. století, přibližně v 60. a 70. letech byla vyvinuta hlavní metoda *backpropagation*, která řídí proces učení. Implementace této metody do neuronových sítí však nebyla jednoduchá a poprvé byla použita v roce 1981, od té doby je využívána stále. Celá historie je velmi obsáhlá a dalo by se říct, že se datuje až do 19. století. Čtenáři, kteří by se chtěli dozvědět více, si mohou přečíst přehled této historie [1]. V současnosti je vývoj velmi aktivní, jsou stále používány základní metody, ale aplikovány na nové architektury sítí s novými vlastnostmi.

2.2 Základní princip

V této sekci si postupně popíšeme základní prvky tvořící neuronovou síť a pokusíme se o vysvětlení principu, jakým tato síť funguje. Tato práce nebude zacházet příliš do detailů, proto kdo by se chtěl dozvědět víc, může následovat literaturu, ze které bylo čerpáno.

Nejprve si popíšeme aktivační funkce, nacházející se skoro v každém neuronu, poté popíšeme tento neuron a jeho roli v celku. Nakonec se budeme snažit vysvětlit, jak se vlastně celá síť "učí", uvedeme dva základní procesy, ke kterým neustále dochází v průběhu učení a ukážeme propojení neuronů v sítí.

2.2.1 Aktivační funkce

Aktivační funkce je důležitou součástí neuronu, která určí, zda byl neuron excitován, neboli aktivován. Excitovaný neuron posílá svůj výstup neuronům, které jsou na něj napojeny. Uvedeme si základní a velmi často používané aktivační funkce [2]. Aktivační funkce se volí podle případu užití. Uvedeme si vzorce pro výpočet aktivačních funkcí a na Obrázku 1 můžeme vidět grafy průběhů některých z nich.

1. Ostrá nelinearita

$$f(x) = \begin{cases} 1 & \text{pokud} \quad x \geq 0 \\ 0 & \text{pokud} \quad x < 0 \end{cases} \tag{1}$$

Tato funkce excituje neuron, pokud vstupní hodnota, překoná určený práh. Ve vzorci je prahovou hodnotou 0. Můžeme si všimnout, že jedinými možnými výstupy jsou 0 a 1, tudíž tato funkce se bude hodit pouze pro binární klasifikace, neboli rozdělení případů do dvou skupin. Pokud bychom se snažili klasifikovat více tříd případů, tak tato funkce není vhodná, protože potřebujeme víc než jen binární výstup.

2. Standardní sigmoida (*Sigmoid*)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Funkce *Sigmoid* řeší hlavní problém ostré nelinearity a to tím, že nabízí nekonečný počet výstupů, všechny v rozmezí od 0 do 1. Je tedy vhodná pro klasifikaci více než dvou tříd. Sigmoid funkce je jedna z nejvíc používaných aktivačních funkcí a hraje důležitou roli ve vícevrstvých sítích [3]. Pokud se podíváme na graf této funkce, můžeme si všimnout jednoho problému, a to toho, že na obou koncích křivky je reakce y na změnu x velmi malá, stále se snižující. Pokud se tedy dostaneme do této části, má za následek velké zpomalení učení. Velikost změny y, neboli derivace funkce, je znázorněna modrou čárkovanou křivkou.

3. Hyperbolický tangens (*TanH*)

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{3}$$

Hyperbolický tangens je velmi podobný *sigmoidě*, také nabízí nekonečný počet výstupů, ale v rozmezí od -1 do 1, což je velmi důležité, pokud chceme aby neurony v další vrstvě mohli také dostávat hodnoty menší než 0. *TanH* má stejný problém se zpomalením učení jako funkce *Sigmoid*. Pokud se ale nepohybujeme na koncích křivky, je reakce y na změnu x větší, než u *Sigmoidy* a to díky záporným hodnotám. Tohoto jevu si můžeme všimnout na modrém čárkovaném grafu derivovaná funkce.

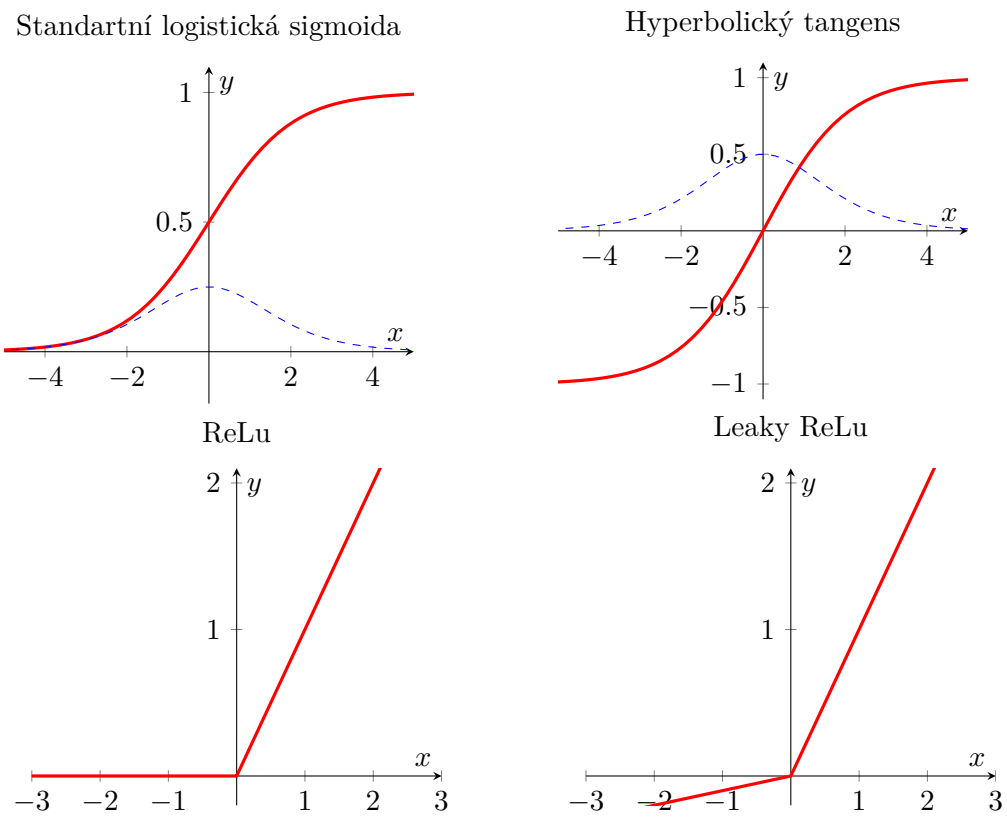
4. ReLu

$$f(x) = \begin{cases} x & \text{pokud} \quad x > 0 \\ 0 & \text{pokud} \quad x \leq 0 \end{cases} \tag{4}$$

Aktivační funkce *ReLu (Rectified Linear Units)* je velmi využívána v konvolučních neuronových sítích. Její výstup je v rozmezí od 0 do ∞. *ReLu* oproti *Sigmoidě* a *TanH* sníží počet aktivací v celé síti, neboť hodnoty ≤ 0 neexcitují neuron, který by mohl excitovat další neurony. Toto je výhodné, neboť méně excitovaných neuronů znamená lehčí síť, menší nároky na výpočty

a několikrát rychlejší trénování [4]. *ReLu* je navíc mnohem jednodušší na výpočet než dříve zmíněné funkce. Jednu nevýhodu ale tato funkce má, během trénování může dojít k "odumření" neuronu, což velmi zjednodušeně znamená, že výstup takového neuronu je pořád 0 a neuron se z tohoto stavu neumí zotavit. Řešením tohoto problému je úprava, která dovoluje malé záporné hodnoty. Tato úprava nemá žádné negativní vlivy na výsledky trénování, pouze dělá neuron robustnější proti zmíněnému problému. Této upravené funkci se říka *Leaky ReLu* [5]. Průběh funkce *Leaky ReLu* můžeme vidět na posledním grafu v Obrázku 1, vzorec poté zde:

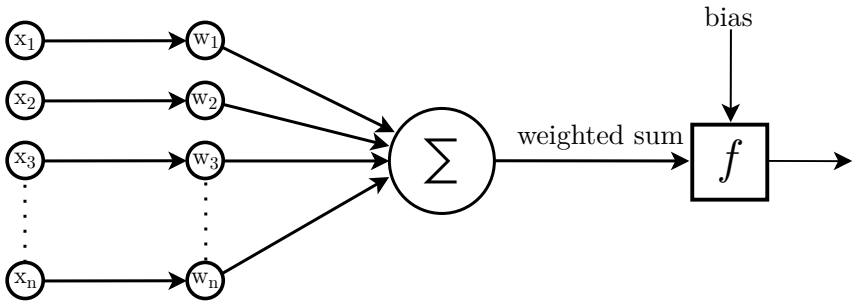
$$f(x) = \begin{cases} x & \text{pokud} \quad x > 0 \\ 0,1\,x & \text{pokud} \quad x \leq 0 \end{cases} \tag{5}$$



Obrázek 1: Grafy základních aktivačních funkcí

2.2.2 Stavba neuronové sítě

Základní stavební jednotkou neuronových sítí je umělý neuron. Tento základní element dále tvoří vrstvy sítě a z těchto vrstev je poskládána celá síť. Umělý neuron byl inspirován tím přírodním ve smyslu, že přijímá několik vstupních signálu (parametrů) a produkuje pouze jeden výsledný aktivační signál. Neurony v neuronových sítích mají navíc pro každý vstupní signál jeden parametr navíc, tímto parametrem je váha (*weight*), která říká jak moc je dané spojení relevantní. Poté, co jsou sečteny vstupní signály podle jejich vah, se k celkové sumě přidá ještě hodnota *bias*, která také řídí aktivaci. Tato hodnota je nezávislá na vstupních datech a je určena učícím algoritmem. Můžeme totiž chtít aby byl neuron aktivován pouze v případě kdy hodnota vážených vstupů je např. 10, poté nastavíme hodnotu *bias* na -10 (*Přesná hodnota nastavení bias záleží na zvolené aktivační funkci.*). Výsledná hodnota vstupuje do aktivační funkce, která určí, zda byl neuron excitován a má poslat svůj výstup dalším napojeným neuronům. Obrázek 2 znázorňuje, jak může vypadat jednoduchý neuron, x_1 až x_n jsou vstupní parametry, w_1 až w_n jsou jejich váhy a f je aktivační funkce. Vzorec (6) ukazuje, jaká hodnota vstupuje do aktivační funkce. Tento neuron by se taky dal považovat za úplně nejjednodušší variantu neuronové sítě s n vstupy a jedním výstupem.



Obrázek 2: Základní model neuronu

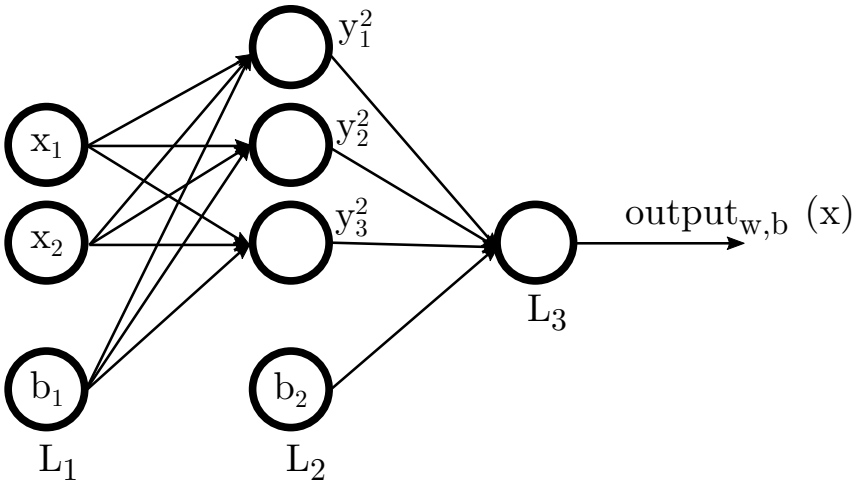
$$activation\ input = \sum_{i=1}^{n}(w_i x_i) + bias \tag{6}$$

Každá neuronová síť má vstupní a výstupní vrstvu, vstupní se stará o data, která přicházejí do sítě a posílá je dalším neuronům v následující vrstvě. Data mohou být různých druhů, záleží pouze na daném způsobu použití sítě. Výstupní vrstva je vždy poslední a produkuje výsledek vzniklý průchodem dat sítí. Mezi těmito nutnými vrstvami může být libovolný počet skrytých vrstev, pokud jich je velké množství, bavíme se o tzv. hlubokých neuronových sítích.

2.2.3 Princip učení

Celý proces učení se skládá ze dvou stále se opakujících operací a to dopředný průchod *forward pass* a zpětný průchod *backpropagation*. Tyto dvě operace se stále opakují, dokud neuronová síť nedosáhne přesnosti, kterou požadujeme, nebo dokud neklesne hodnota *learning rate*, která udává, jak moc se ještě síť učí pod nastavené minimum. Neboť se jedná o "učení pod dohledem" (*supervised learning*), potřebujeme množinu trénovacích dat a také množinu, která říká, jaký výsledek chceme získat po průchodu dat sítí, tuto množinu budeme označovat jako *ground truth*.

2.2.3.1 Dopředný průchod si popíšeme na Obrázku 3, na němž můžeme vidět příklad jednoduché neuronové sítě, která se skládá ze 3 vrstev. (Tato část je inspirována článkem [6])



Obrázek 3: Jednoduchý model neuronové sítě

Písmeny L_l označujeme vrstvy, L_1 je první vrstva, L_2 skrytá a L_3 je poslední vrstva produkující výstup $output_{w,b}(x)$, který je závislý na aktuálních vahách, na hodnotách *bias* a na vstupních datech. Pokud určíme n_l jako počet vrstev, poté můžeme vždy označit poslední vrstvu L_{n_l}. Neurony x_1 a x_2 přijímají vstupní data, které dále posílají do skryté vrstvy na každý napojený neuron. Právě pro tato spojení je důležitá hodnota váhy. Jak jsme mohli vidět na Obrázku 2, potřebujeme hodnotu *bias*, ta je posílána z přidaných jednotek b_l. Váhu, neboli důležitost spojení mezi dvěma jednotkami, budeme označovat jako W_{ij}^l, kde j je neuron ve vrstvě l, a i je neuron ve vrstvě $l+1$. Výstup z neuronu i ve vrstvě l označujeme jako y_i^l, pokud aktivační funkci označíme jako f, můžeme výstupy ze skryté vrstvy a i celkový výstup vypočítat jako:

$$\begin{aligned} y_1^2 &= f(W_{11}^1\ x_1 + W_{12}^1\ x_2 + b_1) \\ y_2^2 &= f(W_{21}^1\ x_1 + W_{22}^1\ x_2 + b_1) \\ y_3^2 &= f(W_{31}^1\ x_1 + W_{32}^1\ x_2 + b_1) \\ output_{w,b}(x) &= f(W_{11}^2\ y_1^2 + W_{12}^2\ y_2^2 + W_{13}^2\ y_3^2 + b_2) \end{aligned} \tag{7}$$

Pokud připomeneme vzorec (6), jeho jednoduchou úpravou dostaneme obecný vzorec (8) na výpočet hodnoty y_i^l. Nutno podotknout, že ve složitějších sítích může mít každý neuron svou hodnotu *bias*, proto zde spodní index u b neznačí vrstvu, ale do kterého neuronu vstupuje. n zde značí počet neuronů v předchozí vrstvě, tedy $l-1$.

$$y_i^l = f\left(\sum_{j=1}^{n}\left(W_{ij}^{l-1}\ y_j^{l-1}\right) + b_i\right) \tag{8}$$

Při implementaci dopředného průchodu se nám hodí vypočítat přechod z jedné vrstvy do druhé, neboli vstupní hodnoty do vrstvy l. Tento výpočet (9) se dá provést pomocí matic, což je velmi výhodné, neboť výpočet pomocí matic je většinou rychlejší, např. díky paralelizaci nebo grafických karet.

$$y^l = f\left(\begin{bmatrix} W_{11}^{l-1} & W_{12}^{l-1} & \dots & W_{1n}^{l-1} \\ W_{21}^{l-1} & W_{22}^{l-1} & \dots & W_{2n}^{l-1} \\ \vdots & \vdots & \ddots & \vdots \\ W_{k1}^{l-1} & W_{k2}^{l-1} & \dots & W_{kn}^{l-1} \end{bmatrix} \begin{bmatrix} y_1^{l-1} \\ y_2^{l-1} \\ \vdots \\ y_n^{l-1} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}\right) \tag{9}$$

2.2.3.2 Zpětný průchod - backpropagation Hlavním úkolem této operace je natrénování, respektive naučení neuronové sítě. Toto učení se provádí úpravou síly spojení (vah) mezi jednotlivými neurony.

Je využíváno ztrátové funkce (tzv. *loss* nebo *cost function*), která má za úkol porovnat hodnoty které jsme získali dopředným průchodem s hodnotami, které chceme získat, neboli *ground truth*. Hodnota této funkce se počítá nad všemi neurony v poslední vrstvě, tudíž její výsledek je závislý na všech hodnotách *weight* a *bias*.

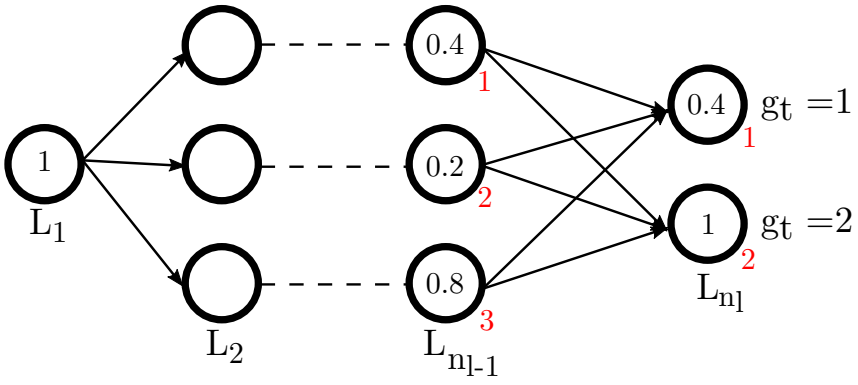
Hodnotu *loss* se snažíme minimalizovat co nejblíž k 0. O tuto minimalizaci se stará metoda *backpropagation*, která upravuje hodnoty *bias* a *weight* u neuronů v sítí. Jedná se o zpětný průchod, tedy informace jsou šířeny od konce sítě na její začátek. Pokud budeme uvažovat velmi jednoduchý případ, kdy trénovací množina má velikost 1 a poslední vrstva se skládá z n neuronů, dá se ztráta vypočítat jako:

$$loss = \sum_{i=1}^{n}(y_n - y_{gt})^2 \tag{10}$$

Kde y_n je hodnota vzniklá průchodem sítě a y_{gt} je *ground truth*. V reálných situacích se bude počítat celková ztráta sítě nad celou trénovací množinou o velikosti m, například následovně:

$$total\ net\ loss = \sum_{i=1}^{m}\sum_{i=1}^{n}(y_n - y_{gt})^2 \tag{11}$$

V této práci nebude odvozen vzorec, pomocí kterého se upravují váhy a hodnoty *bias*, neboť je to nad její rámec, ale bude popsán obecný postup této metody [7]. Jak bylo zmíněno prvním krokem je dopředný průchod prvků trénovací množiny, pomocí kterého získáme na konci výstupní hodnoty, které porovnáme s *ground truth* pomocí již dříve popsané ztrátové funkce. Zbytek práce je již na metodě *backpropagation*, jejímž hlavním úkolem je nalezení minima ztrátové funkce. Jak již bylo řečeno, jedná se o zpětný průchod z poslední vrstvy k první vrstvě, což znamená, že metoda se snaží postupně najít úpravy vah ve vrstvě $L_{n_{l-1}}$, poté $L_{n_{l-2}}$ až L_1, protože vždy potřebuje informace vyšší vrstvy. Jak se váhy upravují názorně ukážeme na Obrázku 4, kde pro zjednodušení vynecháme *bias*, který se upravuje obdobným způsobem. Červená čísla u neuronů jsou jejich označení ve vrstvě, které bude následně použito pro znázornění síly spojení mezi neurony v poslední a předposlední vrstvě v Tabulce 1. Do této sítě vstupuje hodnota 1, kterou bychom také chtěli na konci detekovat na neuronu s *ground truth* $g_t = 1$, hodnota v neuronu by tedy měla být 1.



Obrázek 4: Backpropagation

Můžeme si všimnout, že v poslední vrstvě byl aktivován neuron, který odpovídá hodnotě 2, což je výsledek pro nás nevyhovující. Chtěli bychom aby hodnota v neuronu 2 klesla co

nejblíže k 0 a hodnota v neuronu 1 vzrostla co nejblíže 1. Prvním způsob je pro buňky v poslední vrstvě upravit hodnotu *bias*, což je ale celkem velká změna, která nijak nereflektuje trénovací množinu. Dále můžeme upravit sílu spojení mezi neurony, aktuální síly jsou uvedeny v Tabulce 1. Všimněme si, že nejvýhodnější by bylo, aby hodnota W_{13} vzrostla, protože výstup z třetí buňky předposlední vrstvy je nejsilnější, analogicky bychom chtěli hodnotu W_{23} co nejvíce zmenšit. Úpravy vah se provádějí mezi všemi spojeními, ale tyto dvě jsou uvedeny, protože je vždy nejvýhodnější upravovat spojení, se kterými dosáhneme největší změny. Třetím způsobem a tím nejdůležitějším pro metodu *backpropagation* je změna výstupní hodnoty z předchozí vrstvy vzhledem k síle spojení mezi neurony. Tato úprava se však nedá provést přímo, můžeme si jen zapamatovat, jak se má hodnota změnit. Nutno podotknout, že to jak chceme aby se hodnota změnila neřídí pouze jeden neuron, jako to bylo u vah, ale všechny neurony napojené na neuron, jehož výstup chceme změnit. Poté zprůměrujeme žádosti na změnu od všech buněk a hodnotu si zapamatujeme. Dále se můžeme přesunout do předchozí vrstvy, kde podle žádosti na změnu od vrstvy nadřazené znova provádíme tyto 3 kroky, abychom se dostali co nejblíž k požadované hodnotě. Takto postupujeme až k první vrstvě.

Tabulka 1: Váhy jednotlivých spojení v sítí

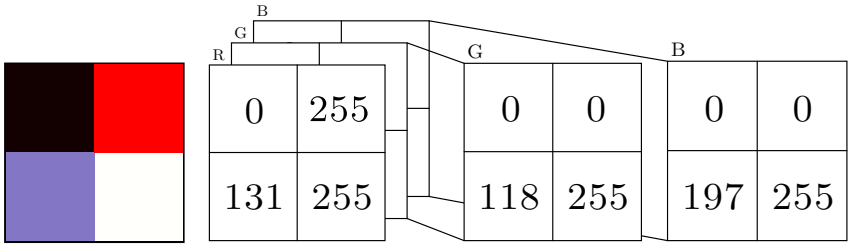
Spojení	Váha
W_{11}	0,50
W_{21}	0,00
W_{12}	0,00
W_{22}	2,00
W_{13}	0,25
W_{23}	0,75

3 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou speciálním druhem neuronových sítí, mají všechny vlastnosti popsány v předchozí sekci 2, rozdíl je ale ve vstupní vrstvě. Tyto sítě přijímají jako vstup obrázky. Úkolem těchto sítí tedy bude nalezení a klasifikace různých objektů v obraze. Pro člověka je tento problém snadno řešitelný. Vezměme si v příklad náš řešený semafor. Člověk ví, že semafor se nachází v určité výšce na sloupu a skládá se, v případě semaforů pro auta, ze 3 světelných kruhů. CNN ale nevidí tyto "vyšší" vlastnosti, umí rozpoznat pouze určité "nižší" vlastnosti, jako jsou hrany a křivky.

3.1 Vstupní data - obrazy

Jak bylo zmíněno vstupními daty jsou obrazy, ty jsou reprezentovány maticemi. V případě barevného obrázku, tedy v barevném prostoru RGB se šířkou w a výškou h, se bude jednat o třídimenzionální matici $w \times h \times 3$, kde hodnoty prvku budou v rozmezí od 0 do 255, maticová reprezentace je znázorněna na Obrázku 5. Obrázek v odstínech šedi, bude reprezentován maticí $w \times h \times 1$, neboť má jen jednu složku barvy.



Obrázek 5: Maticová reprezentace barev

3.2 Konvoluční vrstva

Konvoluční vrstva je nejdůležitější vrstvou v CNN, provádí nejnáročnější výpočty nad vstupními daty a hlavně obsahuje filtry (kernely). Tyto filtry, neboli neurony pro tuto vrstvu, se během procesu učení učí rozpoznávat různé příznaky. Filtrem rozumíme matici, která má stejný počet sloupců jako řádků a musí mít stejný počet dimenzí jako vstupní data, pro barevný obrázek tedy 3. Nejčastější volené filtry mají rozměry 3×3, 5×5 nebo 7×7 řádků, ale volba je specifická pro danou potřebu. Často se uvádí velikost filtru, to je počet sloupců nebo řádků, neboť jsou si rovny. Obecně platí, že menší filtry se budou schopny naučit menší příznaky a větší filtry větší příznaky.

Proces konvoluce je postupné posouvání filtru nad vstupními daty, kdy se při každé zastávce filtru provede skalární součin mezi filtrem a filtrovaným polem. Filtrovaným polem rozumíme data, na kterých se zrovna zastavil filtr. Výsledkem konvoluce je tzv. aktivační mapa, někdy se ji říká mapa příznaků. Znázornění konvoluce můžeme vidět na Obrázku 6, kde pomocí barev-

ných čtverců jsou vyznačena filtrovaná pole, filtr začíná na červeném čtverci, poté se posunuje postupně na modrý, zelený, růžový atd. dokud nedojde do pozice, kdy v levém horním rohu filtrovaného pole bude hodnota 73. Pro zjednodušení uvažujeme pouze dvou–rozměrnou matici a vynecháváme hodnoty *bias*.

7x7x1 vstupní data

10	25	10	77	25
35	85	0	43	12
25	12	73	7	8
36	25	3	9	81
11	14	2	0	0

3x3x1 Filter

0	1	0
-1	1	0
0	1	-1

3x3x1 mapa příznaků

14	-9	119
94	55	-95
13	53	13

Obrázek 6: Konvoluce

Dalšími důležitými parametry, kromě velikosti filtru, jsou *stride* a *zero padding*. *Stride* řídí posun (konvoluci) filtru nad daty, pokud je hodnota 1, posouvá se filtr postupně o 1 dále, jak vidíme na Obrázku 6. Pokud zvýšíme tuto hodnotu, výsledná mapa příznaků bude mít menší rozměry a filtrovaná pole se budou méně překrývat. Pokud bychom tedy pro Obrázek 6 zvětšili *padding* na 2, tak by výsledná mapa příznaků měla rozměry $2 \times 2 \times 1$.

Zero padding, neboli obalení vstupních dat nulami, se nám hodí v případech, kdy nechceme aby výstupní příznaková mapa měla menší rozměry než vstupní data. Jak jsme viděli, po konvoluci dostaneme data o menších rozměrech, než v jakých vstupovali do konvoluce, v CNN můžeme za sebe navázat několik konvolučních vrstev a poté nechceme, aby se stále zmenšovaly rozměry, proto použijeme *zero padding* [8]. Daná hodnota tedy říká, kolik vrstev nul obaluje vstupní data, a chceme–li zachovat rozměry, její hodnotu vypočítáme podle:

$$zero\ padding = \frac{F_l - 1}{2} \tag{12}$$

Kde F_l je velikost filtru. Obecně výpočet výstupního rozměru pro jakoukoliv konvoluční vrstvu, kde I_l je velikost vstupních dat, P je *padding* a S je *stride*, je:

$$out\ size = \frac{I_l - F_l + 2P}{S} + 1 \tag{13}$$

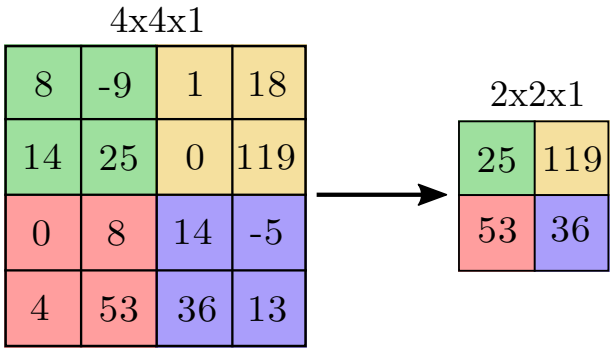
Jak bylo uvedeno na začátku, konvoluční vrstva obsahuje více filtrů, tyto filtry se liší ve svých hodnotách, ale mají stejnou velikost, na konci se sečtou a vznikne výsledná příznaková mapa. Také jsme vypustili vliv parametru *bias*, ten je v konvoluční vrstvě stejný pro všechny filtry a pouze se přičítá k výsledku skalárního součinu.

Jak jsme uvedli v sekci 2.2.3, při zpětném průchodu jsou upravovány hlavně váhy, pro konvoluční vrstvu to jsou hodnoty tvořící filtry. Na začátku procesu učení bude docházet k velkým

změnám ve filtru, což znamená, že síť se bude snažit najít filtr, který bude nejlépe detekovat určitý příznak typický pro detekovaný objekt. Následně, s již menší hodnotou *learning rate*, se bude daný filtr pouze zlepšovat v detekci daného příznaku.

3.3 Pooling

Pokud jsme měli více konvolučních vrstev za sebou, nechtěli jsme, aby docházelo k přílišnému zmenšování rozměru výstupních dat, ale poté, co naše data projdou několika takovými vrstvami, chceme zmenšit rozměry neboli rozlišení. Toto zmenšení se provádí, aby se předešlo *overfittingu*. *Overfitting* je stav, kdy síť nebude schopna rozpoznat prvky jiné než z trénovací množiny. Tedy poté, co je detekován nějaký příznak v obrazu, je méně důležité jeho přesné umístění a více důležité přibližné umístění vzhledem k ostatním příznakům [9]. Pooling má dva parametry a to velikost filtru a *padding*. Nejoblíbenější metodou je *max–pooling*, znázorněný na Obrázku 7.



Obrázek 7: Metoda max–pooling

3.4 Batch normalization

Důležitou technikou používanou v CNN je *batch normalization*. Tento mechanismus představen v článku [10], dovoluje dramatické zrychlení trénování CNN. Principem tohoto mechanismu je úprava vstupů do jakékoliv vrstvy v síti, vstupy upravuje tím způsobem, že je drží kolem nulové hodnoty, což je pro vrstvu, do které vstupují tyto hodnoty výhodné. Tato vrstva má navíc své 2 vlastní parametry, které se také učí. Tyto parametry následně řídí, jestli chceme upravit vstupní hodnoty nebo je ponechat v jejich originální podobě. Název “batch normalization” neboli dávková normalizace je zvolen kvůli faktu, že tato normalizace se neprovádí zvlášť pro každý prvek z trénovací množiny, ale pro dávky o určité velikosti, což také vede k lepší generalizaci naučených parametrů.

4 Detekce semaforů

V této sekci se budeme zabývat samotnou implementací programu, jehož cílem je detekovat semafory v obrazu. Také zmíníme, jak jsme získali datasety a jak musely být připraveny pro potřeby CNN.

K implementaci CNN je použita knihovna Dlib [11], která má již připraveny všechny vrstvy sítě, a nabízí jejich poskládaní dle vlastních potřeb, jinými slovy můžeme vytvořit jakoukoliv architekturu CNN. Velkou výhodou této knihovny je, že je portovatelná a všechny metody, které to dovolí jsou paralelizovány, a to buď na CPU nebo na GPU. Dokonce, pro podporované grafické karty, nabízí Dlib využití metod z knihovny cuDNN [12], která zrychluje standardní procesy v hlubokých neuronových sítích. Pro následnou detekci stavu jsme vyzkoušeli dva různé přístupy, jeden založený na klasických metodách analýzy obrazu a druhý založeny znova na CNN. Pro první klasický způsob byly použity vlastní funkce a pravidla, za pomocí metod z knihovny OpenCV [13]. V přílohách této práce se nachází zdrojový kód, který je zdokumentován systémem Doxygen. Navíc třídy pro trénování a detekci používají C++ *template* systém, takže změna na jiný typ sítě není složitá.

4.1 Datasety

Datasety neboli množina dat, se v našem případě skládá z obrazů získaných pomocí kamery umístěné za čelním sklem automobilu. Zvolená kamera natáčela v rozlišení 1920×1080 při 30 snímcích za sekundu. Získané videa byla nakonec sestříhána na části, kde se nacházejí semafory a následně vyexportována jako sekvence obrázků. Pro potřeby trénování a následného testování, potřebujeme 2 různé datasety M a N takové, jejichž průnik je prázdný, neboli $M \cup N = \emptyset$.

Obrázek 8: Vyznačení semaforu v obrazu

Jelikož se jedná o "učení pod dohledem", musíme ve všech obrázcích v trénovací množině vyznačit semafory, které chceme, ať umí naše síť najít. Přesné označení a nevynechání ani jednoho

semaforu je velmi důležitým krokem kvůli zvolené ztrátové funkci. Navíc máme možnost označit části obrázku jako ignorované, což znamená, že budou naprosto ignorovány v průběhu učení CNN. Ukázku vyznačení semaforů můžeme vidět na Obrázku 8. Tyto anotace semaforu v obrazu jsou vytvořeny pomocí nástroje imglab z knihovny Dlib [11] a následně uloženy v XML souboru, pomocí kterého jsou následně načteny v programu.

Často se pro rozšíření trénovací množiny provádí zrcadlení datasetu, neboli vytvoření zrcadlového obrazu pro všechny obrázky. V našem případě jsme to také zkusili, ale toto rozšíření nevedlo k žádnému zlepšení a pouze trénování trvalo déle. Náš dataset jsme ale rozšířili náhodnými výřezy z obrázků. Tyto náhodné výřezy se generovaly přímo v programu, a následně šly jako vstup do CNN, v tzv. dávkách. Všechny náhodné výřezy musí mít stejnou velikost a hlavně musí obsahovat celý detekovaný objekt. Posledním krokem, ještě než jsou obrázky v dávce poslány do sítě, je narušení barev, což vede ke generalizaci detekovaného objektu. Námi zvolená vstupní vrstva požaduje, aby všechny obrázky z trénovací množiny měly stejné rozměry. Tento požadavek musí být splněn, protože z každého obrázku vytváří pyramidu, která řeší problém velikosti detekovaného objektu. Tato pyramida dovolí detekovat objekt v jakékoliv její úrovni, a tím pádem se může CNN naučit detekovat objekty, které jsou větší nebo menší než objekty nacházející se v trénovací množině. Ukázku takovéto pyramidy můžeme vidět na Obrázku 9, kde vidíme, jak postupně zmenšuje obraz v každé úrovni.

Obrázek 9: Vstupní pyramida

4.2 MMOD ztrátová funkce

MMOD neboli *Max–Margin Object Detection* [14] je metoda pro učení detekce objektů v obrazech. Je založena na principu *sliding window*, neboli okénka, co postupně projíždí obraz. Těchto okének může být i více a pokud máme více různých označených objektů v trénovací množině, tak každý bude mít pro sebe vlastní okénko. To stejné platí i pro různé poměry stran objektů. V našem případě detekujeme pouze jeden druh objektu, ale jsou použity 3 skenovací okénka, neboť se v datasetu nacházejí objekty s třemi různými poměry stran. Tato okénka postupně skenují obraz a v každém kroku jej hodnotí. Toto hodnocení má největší hodnotu v částech obrazu, kde se nachází objekt, který chceme detekovat, proto musí být v obrazu správně označen. Obraz skenovaný těmito okénky můžeme vidět na Obrázku 10, kde červená barva indikuje vysokou možnost výskytu semaforu.

Když se budeme dále v práci zmiňovat o ztrátě sítě, budeme právě myslet ztrátu, která je vypočítána pomocí finální ztrátové vrstvy MMOD. Ztráta v MMOD je vypočítána následně[14]:

$$L(y, y_i) = L_{miss} \cdot (\text{počet chybějících detekcí}) + L_{fa} \cdot (\text{počet falešných detekcí}) \qquad (14)$$

kde y je detekované označení objektu, y_i je skutečné označení tohoto objektu, L_{miss} je ztráta za chybějící detekci, L_{fa} je ztráta za falešnou detekci. Detekovaným a skutečným označením rozumíme všechny detekce a označení v obrazu. Hodnoty obou konstant L_{miss} a L_{fa} jsou rovné 1, tedy hodnotíme stejně chybějící detekci jako falešnou detekci. Podle této hodnoty ztráty jsou poté naše CNN učeny. Teoreticky, pokud bychom chtěli, aby náš detektor detekoval více objektů za cenu více falešných detekcí, můžeme snížit hodnotu L_{fa} a síť bude tedy méně trestána za tyto falešné detekce.

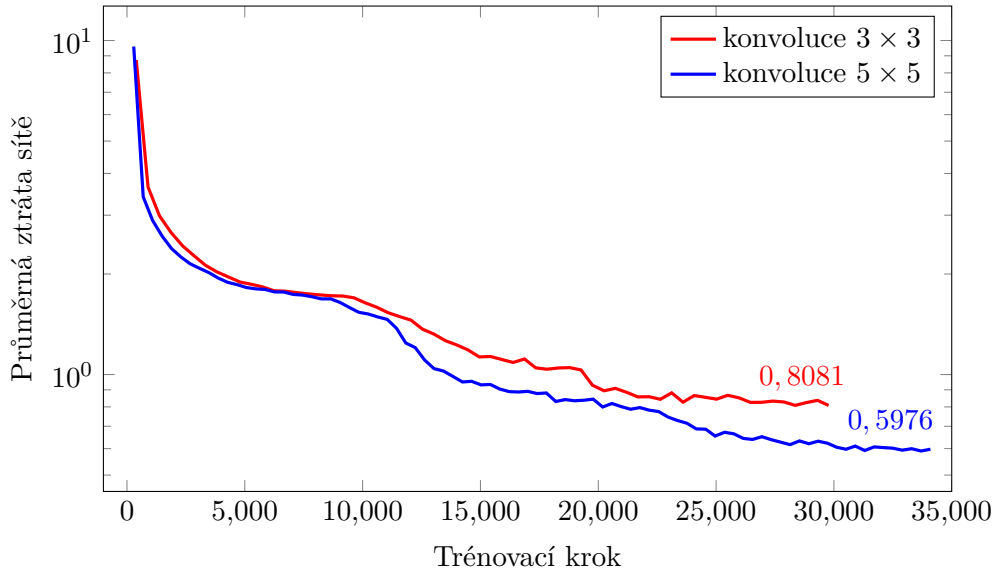
Obrázek 10: Výstup konvoluční neuronové sítě

4.3 Zvolené modely neuronové sítě

Pro detekci semaforu v obraze vyzkoušíme dvě různé CNN. První model je jednodušší, je to neuronová síť z rodiny LeNet [9], tyto sítě jsou lineární a známy již delší dobu. Pro svou jednoduchost využijeme tuto síť i pro detekci stavu semaforu. Druhým modelem bude síť typu ResNet [15], tento druh CNN je novější a také většinou hlubší než sítě typu LeNet.

4.3.1 LeNet

První síť, kterou si popíšeme je typu LeNet [9]. Tato síť je poměrně jednoduchá a v našem případě se skládá ze 24 vrstev. *Learning rate* sítě byl roven $0,1$ a tato hodnota byla zmenšována násobkem $0,1$ poté, co po 2500 trénovacích krocích nedošlo ke zlepšení ztráty sítě. Trénování probíhalo tak dlouho, dokud hodnota *learning rate* byla větší než 10^{-5}. Nejdůležitější operací v CNN je konvoluce a pro naší síť jsme vyzkoušeli konvoluce s velikosti filtru 3×3, 5×5 a 7×7. Při trénování s filtrem 7×7 jsme hned zjistili, že tento filtr je nevhodný na naše data a dále jsme jej nezkoumali. Filtr velikosti 7×7 byl nevhodný, neboť nebyl kompatibilní s námi zvolenou velikosti náhodných výřezů a potřeboval by výřezy o větší velikosti. Zbylé dva filtry jsme vyzkoušeli a natrénovali s nimi síť, výsledky tohoto trénování můžeme vidět v grafu na Obrázku 11. Obě sítě následovaly ze začátku podobný trend, ale síť s konvolucí 5×5 dosáhla menší průměrné ztráty a to $0,5976$, kdežto druhá síť skončila na průměrné ztrátě $0,8081$ a dříve dosáhla minimální hodnoty *learning rate*. Tato data jsou získávána v intervalech poté, co všechny obrázky v dávce prošly sítí.

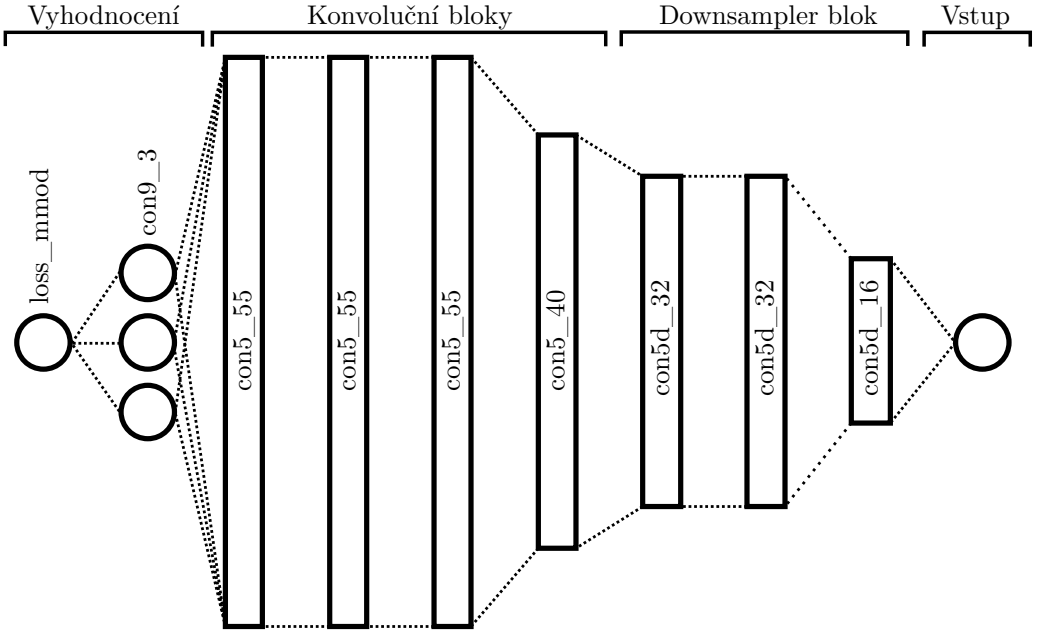


Obrázek 11: Porovnání konvolucí pro síť typu LeNet

Dalším parametrem konvoluce je počet filtrů, který mimo jiné, určuje počet neuronů v konvoluční vrstvě. Počet filtrů jsme zvolili empiricky, v prvních vrstvách jich bylo 16 a postupně jsme jejich počet zvýšili na 55. Posledním parametrem je *stride*, v naší LeNet sítí budeme používat dvě různé konvoluce lišící se právě ve velikosti *stride*. První konvoluce bude mít na výstupu data o stejné velikosti jako na vstupu, a její velikost *stride* je rovná 1. Druhá konvoluce má za úkol data 2× zmenšit a tedy hodnota *stride* je rovna 2. Na tyto konvoluční vrstvy jsou v sítí navázány vrstvy *batch normalization*, které dovolují využití tohoto mechanismu a také vrstva aktivační funkce ReLu, tyto 3 vrstvy budeme nazývat konvolučním blokem.

Vstupní vrstva téhle sítě vytváří pyramidu obrázků, jak jsme mohli vidět na Obrázku 9, tím se několikrát zvětšuje velikost vstupních dat, a proto je třeba data zmenšit. Abychom snížili velikost vstupních dat použijeme *downsampler* blok, který v sobě obsahuje 3 konvoluce, kde každá zmenší velikost dat 2× a tedy celý blok zajistí osminásobné zmenšení vstupních dat.

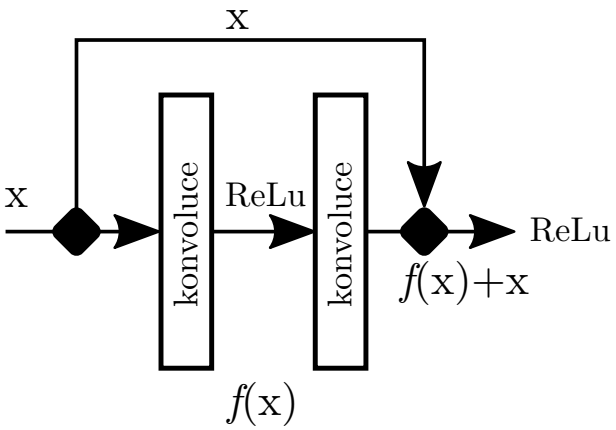
Vstupní vrstva tedy posílá data do *downsampler* bloku, který obsahuje 3 konvoluční bloky, a za tímto blokem následuje 5 konvolučních bloků. Ještě před poslední ztrátovou vrstvou se nachází jedna konvoluce, která je potřebná pro MMOD ztrátovou vrstvu. Jak jsme zmínili, MMOD ztrátová vrstva má určitý počet skenovacích okének, a aby každé okénko dostalo data, je potřeba poslední konvolucí nastavit počet filtru rovný počtu těchto skenovacích okének. Naši LeNet síť můžeme vidět na Obrázku 12. Konvoluční bloky značíme con$X$_Y, kde X je velikost konvoluce $X \times X$ a Y je počet filtrů.



Obrázek 12: Model sítě typu LeNet

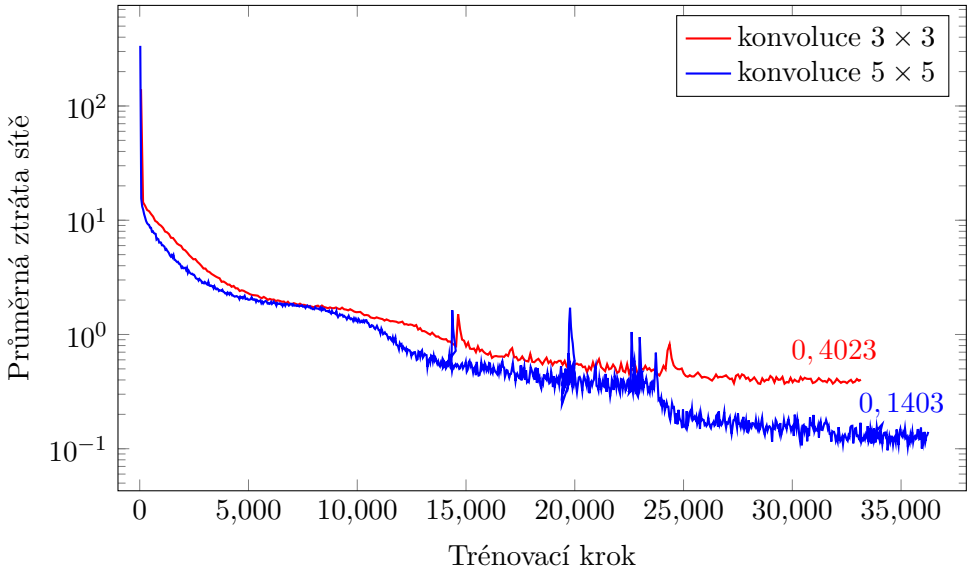
4.3.2 ResNet

Naše druhá síť typu ResNet [15] je mnohem hlubší než první síť a skládá se z 97 vrstev. Hodnota *learning rate* je stejná jako u LeNet sítě a s ní i všechny parametry. Parametry jsme nechali stejné, aby rozdíl výsledků mezi těmito sítěmi byl skutečně způsoben jejich rozdílnou architekturou. ResNet sítě jsou typické tím, že obsahují residuální bloky s tzv. *skip* mechanismem, neboli zkratku mezi vrstvami. Tuto zkratku můžeme vidět znázorněnou na Obrázku 13. Cílem ResNet sítí bylo zjednodušit trénování hlubokých neuronových sítí, což se taky povedlo. Tato optimalizace je právě založena na *skip* mechanismu. Tam, kde je tento mechanismus použit, tedy nejsou výstupem pouze data, která prošla přes konvoluce, ale také nezměněná data. Tedy výstupem z tohoto residuálního bloku je sloučení těchto dvou typů dat $output = f(x) + x$. Pokud se v residuálním bloku provádí zmenšení dat, musí i data procházející přes zkratku být zmenšena na stejnou velikost.



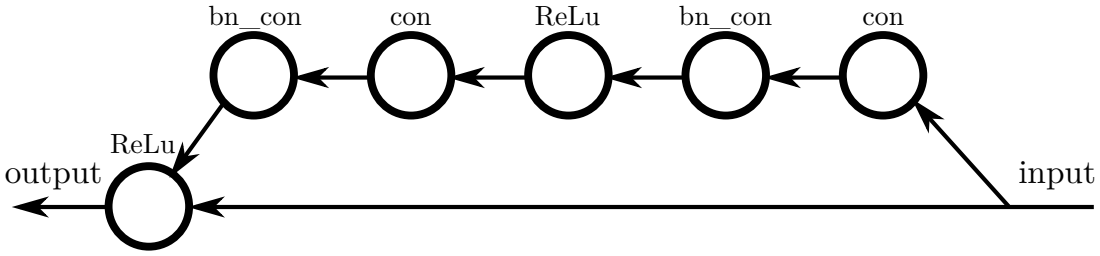
Obrázek 13: Residuální blok

U ResNet sítí jsme také vyzkoušeli dvě různé velikosti konvolucí 3×3 a 5×5, konvoluce 7×7 jsme vynechali z důvodu jejich nekompatibility s našimi daty, která byla zjištěna u LeNet sítě. Jak můžeme vidět na grafu v Obrázku 14, konvoluce s velikostí 5×5 je zase výhodnější, neboť dosáhla menší průměrné ztráty při učení a to $0,1403$. Můžeme si všimnout, že ResNet je v učení více nestabilní, ale nakonec dosahuje lepšího výsledku oproti naší LeNet síti.



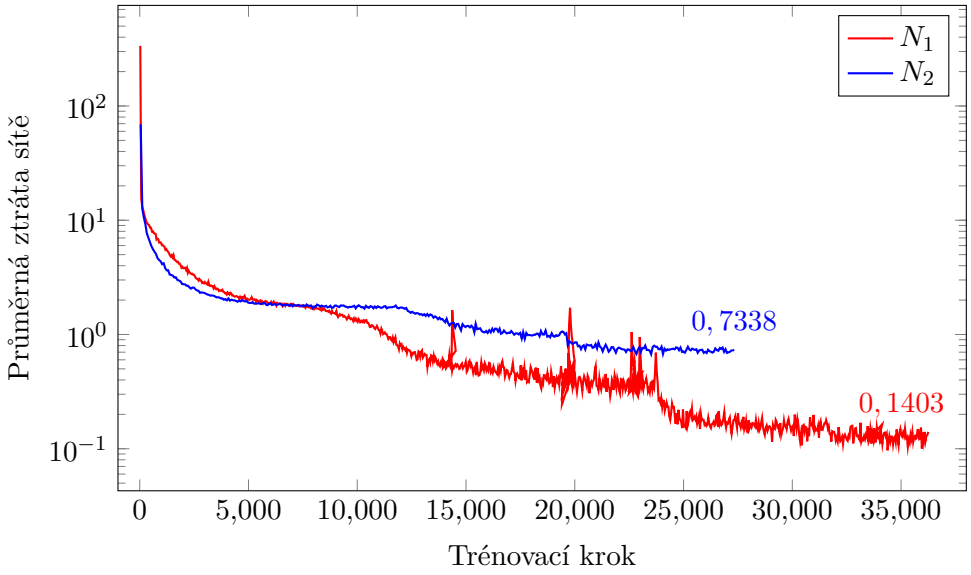
Obrázek 14: Porovnání konvolucí pro síť typu ResNet

Naše ResNet síť zase začíná vstupní vrstvou tvořící pyramidu, po níž však nasleduje pouze jeden residuální blok zmenšující velikost dat, což muže mít za následek menší ztrátu sítě, než kterou jsme dostali u sítě typu LeNet. Poté následují residuální bloky. Tento blok je znázorněn na Obrázku 15, kde con je konvoluce, bn_con je *batch normalization* a ReLu je aktivační funkce ReLu. Mezi residuální bloky je ještě před koncem vložen jeden blok zmenšující velikost, a před ztrátovou vrstvou se zase nachází konvoluce, ze stejného důvodu jak tomu bylo u sítě LeNet.



Obrázek 15: Residuální blok v síti

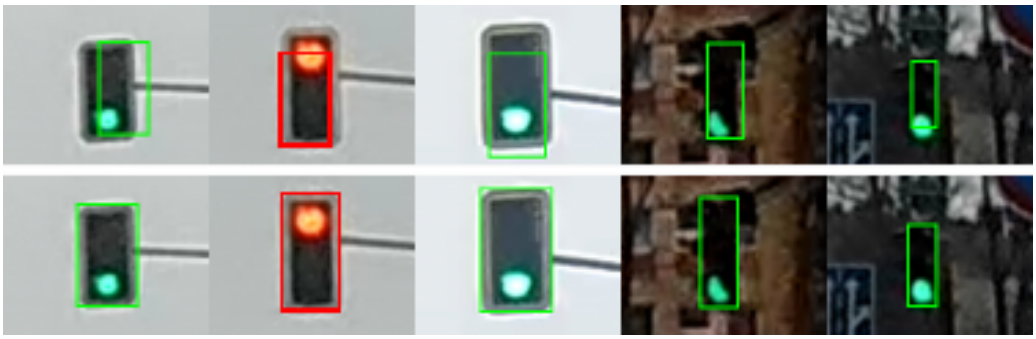
Nevýhodou pouze jednoho residuálního bloku snižující velikost na začátku je velké zpomalení sítě, neboť jí prochází větší objem dat. Proto jsme vyzkoušeli ještě stejnou síť, akorát s jedním tímto blokem snižující velikost dat přidaným na začátek. Originální síť označíme N_1 a tuto novou jako N_2. V grafu na Obrázku 16, můžeme vidět dopad tohoto zmenšení dat na ztrátu sítě, která je razantně větší. Pokud se bavíme o času potřebném k natrénování těchto sítí tak N_1 se trénovala 11 hodin a 8 minut, kdežto N_2 pouze 4 hodiny a 31 minut.



Obrázek 16: Porovnání ResNet sítí v závislosti na velikosti vstupních dat

4.4 *Shape predictor*

Detekovaná pozice semaforu pomocí CNN nemusí být vždy úplně přesná, obdélník může být trochu mimo semafor nebo mít menší či větší rozměry. Ukázku takové situace můžeme vidět v horní polovině Obrázku 17. Vzhledem k tomu, že první způsob detekce stavu (bez CNN) využívá pouze detekovanou část obrazu tj. obdélníkový výřez, tak tato nepřesná detekce zhoršuje následné rozeznání stavu. Řešením této situace je použití *shape predictoru*, který nejenom řeší nepřesné detekce, ale zlepšuje celkovou kvalitu detektoru. Porovnání obou detekcí, bez a s *shape predictorem* můžeme vidět na Obrázku 17, kde spodní polovina ukazuje vylepšenou detekci.



Obrázek 17: Porovnání detekce bez (horní polovina) a s *shape predictorem* (spodní polovina)

Shape predictor je součástí knihovny Dlib [11] a je inspirován článkem [16], který původně řeší problém adjustace orientačních bodů na obličeji v různých úhlech. *Shape predictor* používá pro tuto adjustaci kaskádu regresí, která je natrénována pomocí algoritmu *gradient tree boosting*. Jak můžeme vidět na Obrázku 17 tato metoda se může použít pro obecné zlepšení detektorů. V našem případě používáme pro natrénování kaskády orientační body, které získáme z deteko-

vaných obdélníků na trénovacích datech, přesněji každému obdélníku přiřadíme 5 bodů, 4 body reprezentující strany obdélníku a 1 bod pro jeho střed. Toto rozložení bodů je zvoleno, neboť výsledný *shape predictor* je přesnější než kdyby jsme použili pouze 4 body pro strany obdélníku.

4.5 Rozpoznání stavu semaforu

Rozpoznání stavu je druhý velmi důležitý úkol naší práce. Stav by mohl být rozpoznáván společně s lokací semaforu pomocí naších konvolučních neuronových sítí v jednom kroku, ale kvůli horší kvalitě obrazu jsme se rozhodli rozdělit tyto dvě detekce. Když jsme zkoušeli obě detekce provádět najednou, ztráta sítě byla vysoká a výsledná síť nedokázala dobře detekovat lokaci semaforu. Potřebovali jsme tedy metodu, která jako vstup dostane již nalezený semafor, ve formě obrazu, a určí jeho stav. Nakonec jsme vyzkoušeli dvě takovéto metody, každou založenou na zcela jiném principu.

4.5.1 Detekce stavu analýzou pixelů

První námi navržený způsob využívá základních metod, které se dají provádět s obrazem, a používá některé metody z knihovny OpenCV [13]. Tato detekce, se skládá ze dvou testů. První test, zjišťující průměrný jas v obrazu a druhý test, zjišťující pokrytí obrazu pixely, které leží v určitých hranicích HSV barevného modelu. Vstupní obraz tedy rozdělíme na dva, jeden převedeme do odstínů šedi, pro průměrný jas, a druhý do barevného prostoru HSV.

Než jsme začali používat *shape predictor*, detekce nebyly přesné a stav detekovaný touto metodou byl často zkreslený okolím semaforu, proto jsme se snažili o odstranění tohoto okolí. Okolí bylo odstraňováno pokud průměrný jas na okrajích obrazu byl větší než v jeho centru. Poté byl použit *thresholding*, na což byly nalezeny okraje semaforu a pixely mimo tyto okraje byly nastaveny na nulovou hodnotu. Toto odstranění okolí však bylo velmi nespolehlivé, hlavně kvůli *thresholdingu*, který někdy vyfiltroval samotný semafor.

Následně, tedy se *shape predictorem*, odpadla nutnost odstraňovat okolí a mohli jsme rovnou provádět samotné testy. Tyto testy se prováděli, na třetinách obrazu, tedy vstupní obraz byl rozdělen na 3 části, horní, střední a spodní, každý barevný prostor měl své 3 části. Neboť tyto části jsou na sobě nezávislé, mohli, být tyto testy zparalelizovány na CPU. Paralelizace na GPU nedává smysl kvůli času potřebnému k nakopírování dat do paměti GPU.

Obraz v odstínech šedi je jen matice s hodnotami od 0 do 255, pokud určíme že funkce $I(r, c)$ získá hodnotu pixelu obrázku x, na řádku r a ve sloupci c a obrázek má x_r řádku a x_c sloupců, tak průměrný jas vypočítáme následně:

$$average\ brightness(x) = \frac{\sum_{r=0}^{x_r} \sum_{c=0}^{x_c} I\,(r, c)}{x_r \cdot x_c} \tag{15}$$

Pro zlepšení tohoto testu jsme ještě před výpočtem průměrného jasu hodnoty pixelu normalizovali do intervalu od 0 do 95, čímž jsme odstranili šum.

HSV test používá metodu *inRange*, tato metoda kontroluje zda se pixel nachází v námi definovaném intervalu HSV hodnot, pokud se pixel nevejde do tohoto intervalu je nastaven na 0. Pro každou část semaforu definujeme jiný interval barev a každé části přidáváme interval pro bílou barvu, neboť velmi jasné barvy přecházejí do bílé. Výstupem *inRange* jsou pixely s hodnotou 255, odpovídající pixelům spadajícím do daného intervalu. Jelikož používáme proměnný počet intervalů, musíme všechny masky binárně sečteme. Následně zjišťujeme pokrytí obrazu maskou, pokrytím rozumíme poměr bílých pixelů s hodnotou 255 ke všem pixelům v obrazu.

Následně podle našich pravidel určíme detekovaný stav. Pro první test, průměrného jasu, nás zajímá část obrazu s největším průměrným jasem. Pro HSV test je důležité co největší pokrytí maskou. Pokud pro všechny části v HSV testu zjistíme nulové pokrytí, stav semaforu je neaktivní. Pokud se část s největším průměrným jasem rovná části s největším pokrytím HSV maskou, detekujeme stav náležící dané části, tedy červená pro horní část, oranžová pro prostřední část a zelená pro spodní část. Tato metoda je založena na práci s hodnotou pixelu, na našich pravidlech a pro obraz horší kvality se ukázala méně spolehlivou, což byla motivace k vyzkoušení další metody.

4.5.2 Detekce stavu pomocí CNN

Druhá metoda tedy zase využívá CNN, přesněji síť typu LeNet. Jelikož chceme, aby detekce stavu byla co nejrychlejší tak i tuto síť jsme navrhli tak, aby průchod daty přes ni byl co nejrychlejší. Proto použijeme jen 2 konvoluční bloky s 40 a 55 filtry. Data potřebná pro natrénování této sítě jsme vzali ze stejné trénovací množiny, pomocí které jsme natrénovali naše sítě na detekci semaforu. Následně jsme si uložili detekované semafory a tím vytvořili novou trénovací množinu, která obsahuje přes 1300 prvků V tomto případě by se dala použít CNN, která nepotřebuje v trénovacích datech vyznačené lokace objektů, ale určila by stav podle celého obrazu. My jsme avšak vyznačili lokace objektů, kterými v tomto případě jsou svítící části semaforu tedy kružnice nebo šipky. Pomocí těchto detekcí následně určujeme stav semaforu. Ve většině případů svítí na semaforu jedná část, poté stav určujeme podle horní hranice detekce následovně:

```
//image.nr() odpovídá výšce detekovaného semaforu.
long height28Perc = (long)(0.28f * image.nr());
long height55Perc = (long)(0.55f * image.nr());
long height60Perc = (long)(0.6f * image.nr());

if (detectionRect.top() < height28Perc)
   return Red;
if (detectionRect.top() > height28Perc && detectionRect.top() < height55Perc)
   return Orange;
if (detectionRect.top() > height60Perc)
   return Green;
```

V případě dvou detekcí, přechod semaforu z červené na zelenou, vyhodnotíme každou detekci zvlášť a pokud je jedna z nich určena jako červená a druhá jako oranžová, detekujeme stav oranžová.

4.6 Testování na reálných datech

V závěrečném testování na reálných datech spolu porovnáme dvě sítě, které vyšly jako nejlepší pro každou architekturu tedy LeNet a ResNet, obě s konvolucí 5×5. Testovací dataset se skládá z 200 obrázků, ve kterých se nachází 628 detekovatelných objektů. Zastoupení jednotlivých stavů můžeme vidět v Tabulce 2. Testování bylo prováděno na 1 grafické kartě NVidia GeForce 1070, která má 1920 CUDA jader.

Tabulka 2: Zastoupení stavů v testovacím datasetu

Stav	Počet výskytů
Červená	291
Oranžová	133
Zelená	204

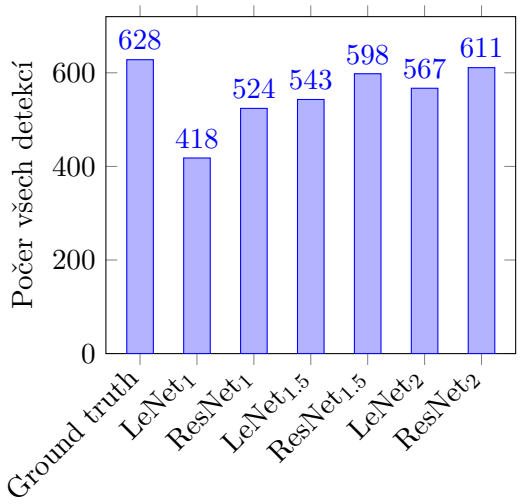
4.6.1 Test detekce lokace semaforu

Při hodnocení detekce lokace, budeme hodnotit přesnost (*precision*) a senzitivitu (*recall*) jednotlivých detektorů a následně obě hodnoty použijeme pro výpočet F1 skóre. F1 skóre nám dá výsledné skóre detektoru s nejlepší možnou hodnotou 1 a nejhorší 0. Všechny tyto hodnoty jsou vypočteny následovně:

$$\begin{aligned} precision &= \frac{\text{počet pozitivních detekcí}}{\text{počet všech detekcí}} \\ recall &= \frac{\text{počet pozitivních detekcí}}{\text{všechny správné detekce}} \\ \text{F1} &= 2 \cdot \frac{precision \cdot recall}{precision + recall} \end{aligned} \tag{16}$$

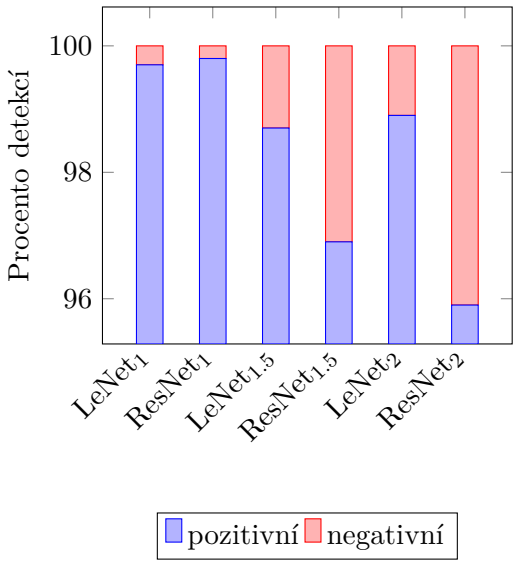
Dané ohodnocení jsme také prováděli pro různé velikosti vstupních dat, a proto se může v následujících grafech a tabulkách objevit název sítě se spodním indexem, znázorňujícím faktor, kterým byla vstupní data zvětšena, např. $\text{LeNet}_{1.5}$ je síť typu LeNet a vstupní data byla $1.5\times$ zvětšena.

Nejprve se podíváme na to, kolik byly detektory schopné detekovat objektů v závislosti na velikosti vstupních dat. Toto můžeme vidět na grafu v Obrázku 18. Obecně platí, že s větší velikostí vstupních dat naše sítě detekují více objektů, dále pak vidíme, že detektory využívající ResNet síť jsou schopny detekovat více objektů při stejné velikosti dat. Toto bude způsobeno tím, že obsahují pouze jeden blok snižující velikost dat po vstupní vrstvě.



Obrázek 18: Počet detekovaných objektů

Se vzrůstajícím počtem detekcí, ale také vzrůstá počet falešných detekcí, toto je pravda hlavně pro ResNet detektory. Tento trend můžeme vidět na grafu v Obrázku 19, který porovnává poměr pozitivních a negativních detekcí. Nutno říci, že i v nejhorším případě je počet negativních detekcí menší než 5 % ze všech detekcí.



Obrázek 19: Poměr pozitivních a negativních detekcí

Přesnost, citlivost, *F1 skóre* detektoru a další výsledky můžeme vidět v Tabulce 3. Všimněme si, že větší velikost vstupních dat vede k lepším výsledkům, jedinou výjimkou je tomu u detektoru ResNet_2. Tento detektor nedosáhl lepšího skóre právě kvůli negativním detekcím, které jsme si ukázali v předchozím grafu na Obrázku 19. Pokud bychom se chtěli bavit o nejlepším detektoru, byl by to zřejmě LeNet_2, jehož skóre sice není tak vysoké jako u detektorů s ResNet sítí, ale čas potřebný pro průchod jednoho obrázku konvoluční sítí je mnohem menší. Větší hodnoty zvětšení jsme nezkoušeli, neboť paměťová a časová složitost by byla příliš velká. Ukázku detekce, i se stavem semaforu, můžeme vidět na Obrázku 20, další pak v Příloze B této práce nebo v elektronických přílohách.

Tabulka 3: Výsledky testu detekce lokace semaforu

Detektor	Přesnost	Citlivost	F1 skóre	Průchod obrázku sítí[ms]
LeNet_1	0,9976	0,6640	0,7973	78,0650
$\text{LeNet}_{1.5}$	0,9871	0,8535	0,9155	168,9150
LeNet_2	**0,9894**	**0,8933**	**0,9389**	**298,3200**
ResNet_1	0,9981	0,8328	0,9080	233,6450
$\text{ResNet}_{1.5}$	**0,9699**	**0,9236**	**0,9462**	**509,4000**
ResNet_2	0,9591	0,9331	0,9460	898,3300

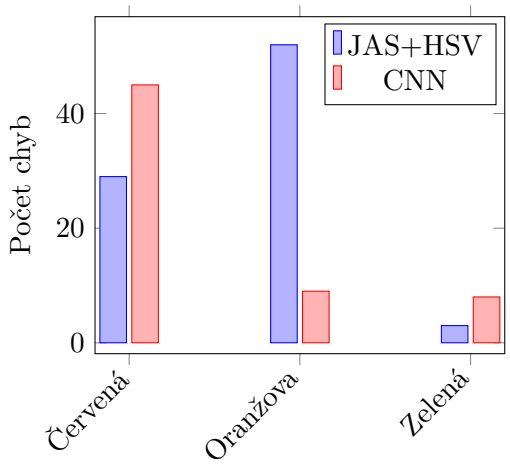
Obrázek 20: Ukázka detekce semaforů

4.6.2 Test detekce stavu semaforu

Přesnost detekce stavu hodnotíme počtem chyb. V Tabulce 4 můžeme vidět výsledky obou naších vyzkoušených metod. Obě metody, pro nás překvapivě, dosahují podobné přesnosti. Metoda Jas+HSV sice dosahuje větší rychlosti zpracování semaforu, ale ani u CNN metody není doba pro zpracování příliš dlouhá. Pro metodu CNN a zvětšení 2× je průměrná doba zpracování 2,5 milisekundy. Pokud se podíváme podrobněji na chyby v detekci, které můžeme vidět na grafu v Obrázku 21, pro nejlepší kombinaci metody a zvětšení, zjistíme, že první metoda Jas+HSV se zvětšením 1× není konzistentní pro žádnou barvu a chyby v detekci se tedy nacházejí nad všemi stavy semaforu. Oproti tomu metoda CNN se zvětšením 2× ve většině případů nedokázala detekovat červenou barvu (stav stop). Z tohoto můžeme usoudit, že dataset, který jsme využili pro natrénování této sítě nebyl dostatečně vyvážený. Tímto rozumíme, že jsme nebyli schopni dostatečně generalizovat detekovaný objekt, což vedlo k nemožnosti detekovat některé případy. Detekce pomocí CNN by se tedy dala poměrně jednoduše zlepšit vhodnějším datasetem a proto ji považujeme za lepší způsob řešení daného problému.

Tabulka 4: Výsledky testu detekce stavu semaforu

Metoda	Zvětšení dat	Počet chyb	Přesnost [%]
Jas+HSV	1	67	89,3312
Jas+HSV	1,5	82	86,9426
Jas+HSV	2	103	83,5987
CNN	1	175	72,1337
CNN	1,5	91	85,5095
CNN	2	64	89,8089



Obrázek 21: Chyby v detekci stavu podle barvy

5 Závěr

V této práci jsme zjednodušeně popsali funkčnost neuronových sítí, princip učení a toky dat v síti. Dále jsme se zaměřili na konvoluční neuronové sítě, které jsme zvolili jako řešení pro náš problém, detekci semaforu v obrazech. Popsali jsme princip CNN, uvedli jsme, jaké specifické vrstvy tyto sítě obsahují a jak fungují. Pro řešení lokace semaforu v obrazu jsme porovnali dva typy CNN, každý založený na jiné architektuře. Také jsme popsali stavbu těchto sítí a natrénovali jsme je s daty, které jsme pro tento účel získali. Pro detekci stavu semaforu jsme představili dvě zcela rozdílné metody, které nakonec nečekaně fungovaly s podobnou přesností, i když metodu využívající CNN hodnotíme jako lepší, z důvodu snadnější možnosti vylepšení. Naše detektory jsme otestovaly na datech získaných z kamery, která byla umístěné ve vozidle. Tyto výsledky jsme porovnali a ohodnotili pomocí F1 skóre. Detekci jsme navíc vyzkoušeli pro různé velikosti vstupních dat. Jsme si vědomi, že námi dosažené výsledky, by nejspíše mohly být ještě vylepšeny kvalitnějšími datasety, obsahující obrazy lepší kvality. V úvodu jsme zmínili použití v autonomních vozidlech. Náš detektor ještě nedosahuje takových kvalit, aby mohl být použit v takovémto vozidle. Navíc naše data jsou omezena pouze na denní dobu, z důvodu neschopnosti kamery zachytit kvalitní obraz v noci. V budoucí práci bychom chtěli vylepšit kvalitu detekce lokace semaforu, natrénovat CNN pro detekci stavu vhodnějším datasetem a vyzkoušet další moderní architektury konvolučních sítí. V poslední řadě bychom rádi rozšířili dataset na data z večerních a nočních hodin, čímž bychom docílili lepší použitelnosti systému.

Literatura

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[2] E. Volná, "Neuronové sítě 1," *Ostrava: Ostravská univerzita v Ostravě. Vydání: druhé*, 2008.

[3] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[5] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[6] "Multi-Layer Neural Network." `http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/`. [Online; 8. února 2018].

[7] I. Vondrák and V. škola báňská Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky, *Umělá inteligence a neuronové sítě*. VŠB - Technická univerzita Ostrava, 2009.

[8] "CS231n Convolutional Neural Networks for Visual Recognition." `http://cs231n.github.io/convolutional-networks/`. [Online; 2. března 2018].

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[11] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[12] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[13] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

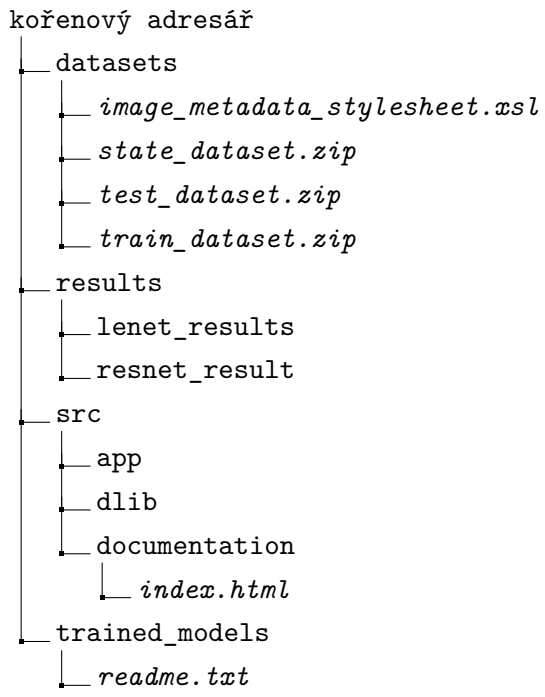
[14] D. E. King, "Max-margin object detection," *CoRR*, vol. abs/1502.00046, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[16] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1867–1874, June 2014.

A Obsah přiloženého DVD

A.1 Adresářová struktura



A.2 Složka datasets

Tato složka obsahuje "zazipované" adresáře a jeden soubor dovolující zobrazení datasetů v internetovém prohlížeči. Soubor *state_dataset.zip* obsahuje obrázky k natrénování CNN k detekci stavu semaforu, *train_dataset.zip* pak obrázky k natrénování CNN k detekci lokace semaforu. Testování bylo prováděno nad datasetem nacházejícím se v *test_dataset.zip*.

A.3 Složka results

V podsložkách tohoto adresáře najdeme obrázky s detekovanými semafory z testovacího datasetu. Podsložky začínají názvem sítě, která byla využita k jejich detekci.

A.4 Složka src

Tento adresář obsahuje zdroje knihovny Dlib a našeho programu. Dále se zde ve složce documentation nachází programátorská dokumentace, která se spouští souborem *index.html*.

A.5 Složka trained_models

Zde najdeme natrénované modely sítí. Další podrobnosti, který soubor patří k jaké síti najdeme v souboru *readme.txt*.

B Ukázky detekce